# Lecture 5 (Part 1)
# Chapter4

Topics covered:
Input/Output Organization

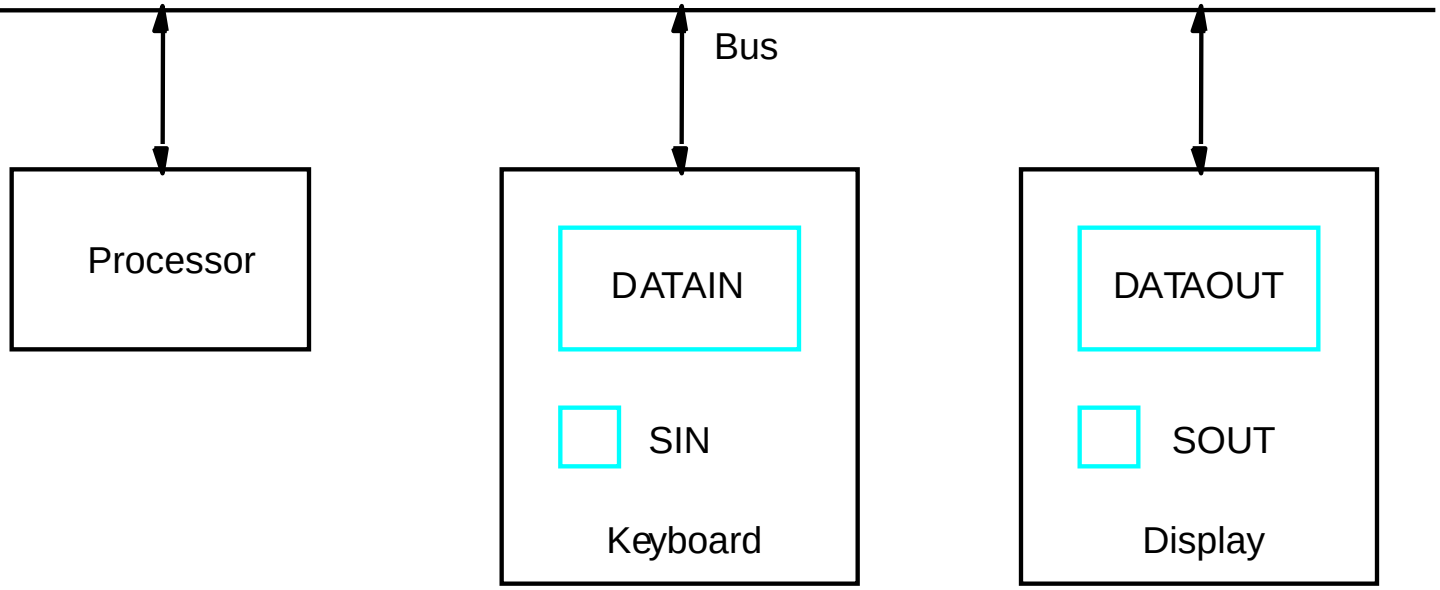
# Basic Input/Output Operations

- We have seen instructions to:
  - Transfer information between the processor and the memory.
  - Perform arithmetic and logic operations
  - Program sequencing and flow control.
- Input/Output operations which transfer data from the processor or memory to and from the real world are essential.
- In general, the rate of transfer from any input device to the processor, or from the processor to any output device is likely to the slower than the speed of a processor.
  - The difference in speed makes it necessary to create mechanisms to synchronize the data transfer between them.

# Basic Input/Output operations (contd..)

- Let us consider a simple task of reading a character from a keyboard and displaying that character on a display screen.
- *A simple way of performing the task is called program-controlled I/O.*
- There are two separate blocks of instructions in the I/O program that perform this task:
  - One block of instructions transfers the character into the processor.
  - Another block of instructions causes the character to be displayed.

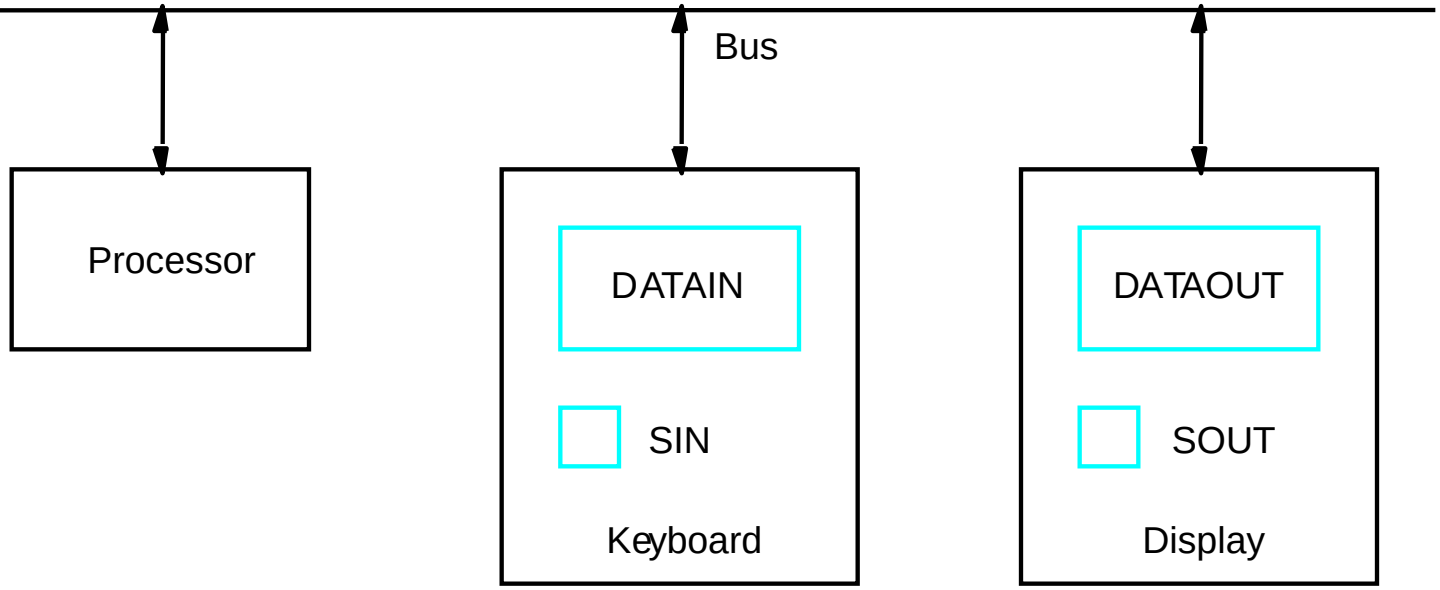
# Basic Input/Output operations (contd..)




*Input:*

- *When a key is struck on the keyboard, an 8-bit character code is stored in the buffer register DATAIN.*
- *A status control flag SIN is set to 1 to indicate that a valid character is in DATAIN.*
- *A program monitors SIN, and when SIN is set to 1, it reads the contents of DATAIN.*
- *When the character is transferred to the processor, SIN is automatically cleared.*
- *Initial state of SIN is 0.*

# Basic Input/Output operations (contd..)




*Output:*

- *When SOUT is equal to 1, the display is ready to receive a character.*
- *A program monitors SOUT, and when SOUT is set to 1, the processor transfers a character code to the buffer DATAOUT.*
- *Transfer of a character code to DATAOUT clears SOUT to 0.*
- *Initial state of SOUT is 1.*

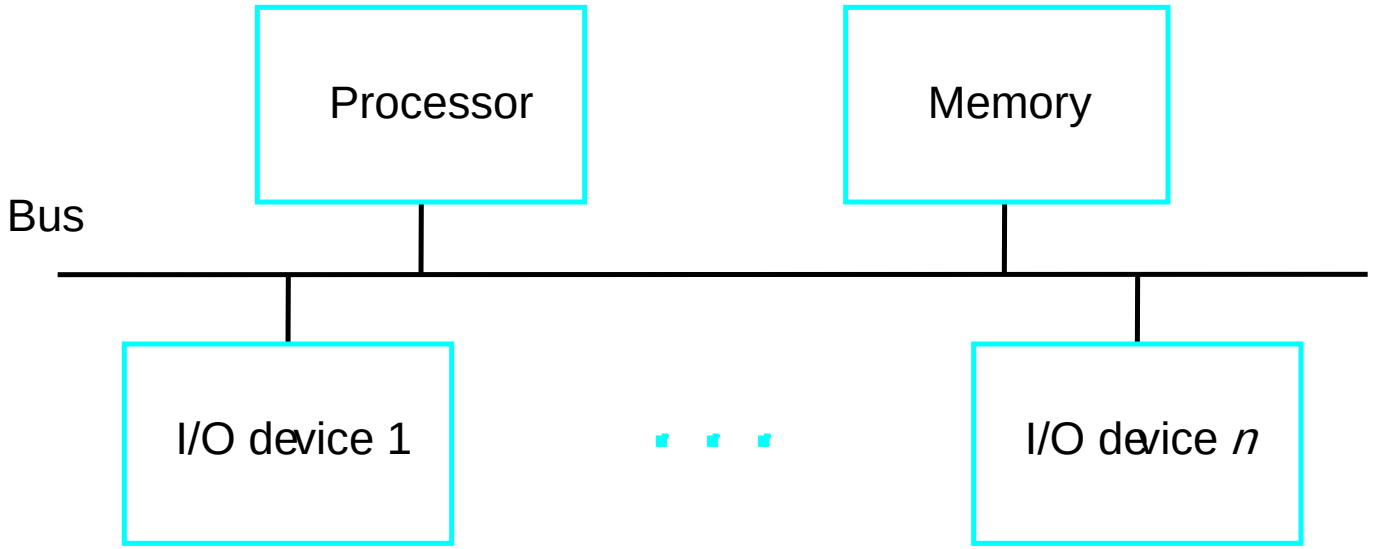
# Basic Input/Output operations (contd..)

- Buffer registers DATAIN and DATAOUT, and status flags SIN and SOUT are part of a circuitry known as device interface.
- To perform I/O transfers, we need machine instructions to:
  - Monitor the status of the status registers.
  - Transfer data among the I/O devices and the processor.
- Instructions have similar format to the instructions used for moving data between the processor and the memory.
- How does the processor address the I/O devices?
  - Some memory address values may be used to refer to peripheral device buffer registers such as DATAIN and DATAOUT.
  - This arrangement is called as Memory-Mapped I/O.

```
Move    DATAIN,R0
```

```
Move    R0,DATAOUT
```

# Accessing I/O devices

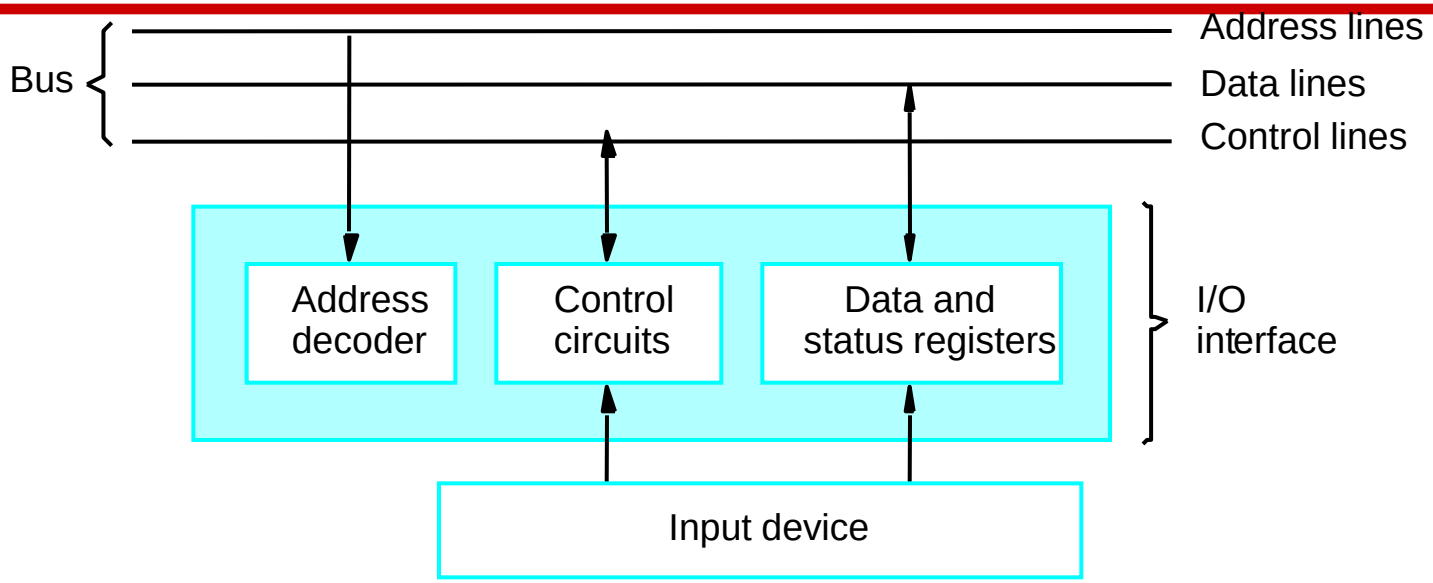



- *Multiple I/O devices may be connected to the processor and the memory via a bus.*
- *Bus consists of three sets of lines to carry address, data and control signals.*
- *Each I/O device is assigned an unique address.*
- *To access an I/O device, the processor places the address on the address lines.*
- *The device recognizes the address, and responds to the control signals.*

# Accessing I/O devices (contd..)

- I/O devices and the memory may share the same address space:
  - Memory-mapped I/O. (Intel processors)
  - Any machine instruction that can access memory can be used to transfer data to or from an I/O device.(Simpler software)
  - uses the same address space.
  - This increases the cost of adding hardware to the machine.
- I/O devices and the memory may have different address spaces:
  - Port-mapped I/O.(Motorola processors)
  - Special instructions to transfer data to and from I/O devices.
  - uses a separate, dedicated address space.
  - I/O devices deals with fewer address lines.
  - less cost to add hardware devices to a machine.
  - I/O address lines need not be physically separate from memory address lines.
  - In fact, address lines may be shared between I/O devices and memory, with a control signal to indicate whether it is a memory address or an I/O address.

# Hardware required to connect an I/O device to the bus




- *I/O device is connected to the bus using an I/O interface circuit which has:*
  - *Address decoder, control circuit, and data and status registers.*
- *Address decoder decodes the address placed on the address lines thus enabling the device to recognize its address.*
- *Data register holds the data being transferred to or from the processor.*
- *Status register holds information necessary for the operation of the I/O device.*
- *Data and status registers are connected to the data lines, and have unique addresses.*
- *I/O interface circuit coordinates I/O transfers.*

# Accessing I/O devices (contd..)

- Recall that the rate of transfer to and from I/O devices is slower than the speed of the processor. This creates the need for mechanisms to synchronize data transfers between them.
- Program-controlled I/O:
  - Processor repeatedly monitors a status flag to achieve the necessary synchronization.
  - Processor polls the I/O device.
- Two other mechanisms used for synchronizing data transfers between the processor and memory:
  - Interrupts.
  - Direct Memory Access.

# Program controlled I/O Example

The program in Figure 4.4 is similar to that in Figure 2.20. This program reads a line of characters from the keyboard and stores it in a memory buffer starting at location LINE. Then, it calls a subroutine PROCESS to process the input line. As each character is read, it is *echoed back* to the display. Register R0 is used as a pointer to the memory buffer area. The contents of R0 are updated using the Autoincrement addressing mode so that successive characters are stored in successive memory locations.

Each character is checked to see if it is the Carriage Return (CR) character, which has the ASCII code 0D (hex). If it is, a Line Feed character (ASCII code 0A) is sent to move the cursor one line down on the display and subroutine PROCESS is called. Otherwise, the program loops back to wait for another character from the keyboard.

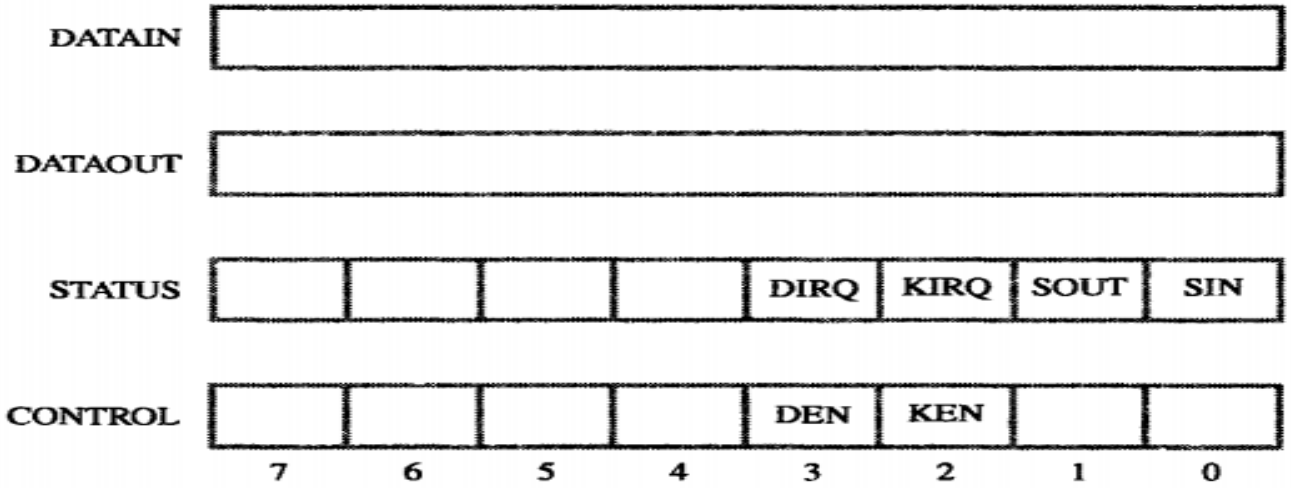



**Figure 4.3** Registers in keyboard and display interfaces.

# Program controlled I/O Example

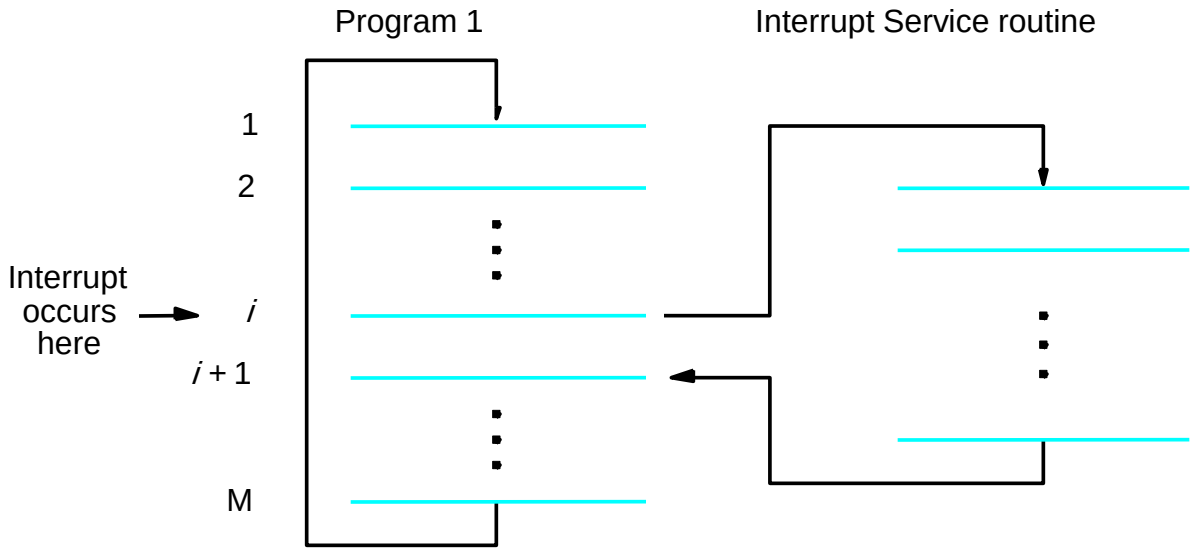
```
        Move        #LINE,R0          Initialize memory pointer.
WAITK   TestBit     #0,STATUS         Test SIN.
        Branch=0    WAITK             Wait for character to be entered.
        Move        DATAIN,R1         Read character.
WAITD   TestBit     #1,STATUS         Test SOUT.
        Branch=0    WAITD             Wait for display to become ready.
        Move        R1,DATAOUT        Send character to display.
        Move        R1,(R0)+          Store charater and advance pointer.
        Compare     #$0D,R1           Check if Carriage Return.
        Branch≠0    WAITK             If not, get another character.
        Move        #$0A,DATAOUT      Otherwise, send Line Feed.
        Call        PROCESS           Call a subroutine to process the
                                        the input line.
```

**Figure 4.4** A program that reads one line from the keyboard, stores it in memory buffer, and echoes it back to the display.

# Interrupts

- In program-controlled I/O, when the processor continuously monitors the status of the device, it does not perform any useful tasks.
- An alternate approach would be for the I/O device to alert the processor when it becomes ready.
  - Do so by sending a hardware signal called an interrupt to the processor.
  - At least one of the bus control lines, called an interrupt-request line is dedicated for this purpose.
- Processor can perform other useful tasks while it is waiting for the device to be ready.

# Interrupts (contd..)




- *Processor is executing the instruction located at address i when an interrupt occurs.*
- *Routine executed in response to an interrupt request is called the interrupt-service routine.*
- *When an interrupt occurs, control must be transferred to the interrupt service routine.*
- *But before transferring control, the current contents of the PC (i+1), must be saved in a known location.*
- *This will enable the return-from-interrupt instruction to resume execution at i+1.*
- *Return address, or the contents of the PC are usually stored on the processor stack.*

# Interrupts (contd..)

- Treatment of an interrupt-service routine is very similar to that of a subroutine.
- However there are significant differences:
  - A subroutine performs a task that is required by the calling program.
  - Interrupt-service routine may not have anything in common with the program it interrupts.
  - Interrupt-service routine and the program that it interrupts may belong to different users.
  - As a result, before branching to the interrupt-service routine, not only the PC, but other information such as condition code flags, and processor registers used by both the interrupted program and the interrupt service routine must be stored.
  - This will enable the interrupted program to resume execution upon return from interrupt service routine.

# Interrupts (contd..)

- Saving and restoring information can be done automatically by the processor or explicitly by program instructions.
- Saving and restoring registers involves memory transfers:
  - Increases the total execution time.
  - Increases the delay between the time an interrupt request is received, and the start of execution of the interrupt-service routine. This delay is called interrupt latency.
- In order to reduce the interrupt latency, most processors save only the minimal amount of information:
  - This minimal amount of information includes Program Counter and processor status registers.
- Any additional information that must be saved, must be saved explicitly by the program instructions at the beginning of the interrupt service routine.

# Interrupts (contd..)

- When a processor receives an interrupt-request, it must branch to the interrupt service routine.
- It must also inform the device that it has recognized the interrupt request.
- This can be accomplished in two ways:
  - Some processors have an explicit interrupt-acknowledge control signal for this purpose.
  - In other cases, the data transfer that takes place between the device and the processor can be used to inform the device.

# Interrupt Hardware

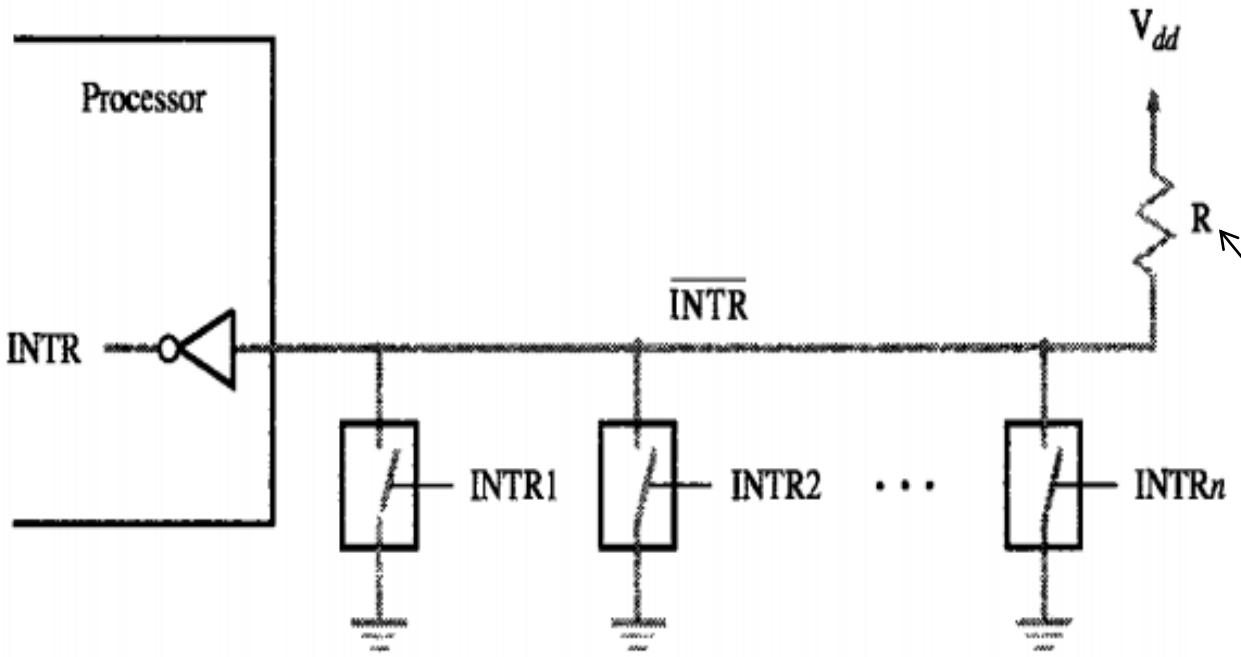



**Figure 4.6** An equivalent circuit for an open-drain bus used to implement a common interrupt-request line.

The circuit used to drive INTR line is implemented as open-collector for bipolar circuits or open-drain for MOS circuit.

# Interrupts (contd..)

- Interrupt-requests interrupt the execution of a program, and may alter the intended sequence of events:
  - Sometimes such alterations may be undesirable, and must not be allowed.
  - For example, the processor may not want to be interrupted by the same device while executing its interrupt-service routine.
- Processors generally provide the ability to enable and disable such interruptions as desired.
- One simple way is to provide machine instructions such as *Interrupt-enable* and *Interrupt-disable* for this purpose.
- To avoid interruption by the same device during the execution of an interrupt service routine:
  - First instruction of an interrupt service routine can be Interrupt-disable.
  - Last instruction of an interrupt service routine can be Interrupt-enable.

# The sequence of events involved in handling an interrupt

1. The device raises an interrupt request.
2. The processor interrupts the program currently being executed.
3. Interrupts are disabled by changing the control bits in the PS
4. The device is informed that its request has been recognized, and in response, it deactivates the interrupt-request signal.
5. The action requested by the interrupt is performed by the interrupt-service routine.
6. Interrupts are enabled and execution of the interrupted program is resumed.

# Handling Multiple I/O devices

- Multiple I/O devices may be connected to the processor and the memory via a bus. Some or all of these devices may be capable of generating interrupt requests.
  - Each device operates independently, and hence no definite order can be imposed on how the devices generate interrupt requests?
- How does the processor know which device has generated an interrupt?
- How does the processor know which interrupt service routine needs to be executed?
- When the processor is executing an interrupt service routine for one device, can other device interrupt the processor?
- If two interrupt-requests are received simultaneously, then how to break the tie?

# Handling Multiple I/O devices

- Consider a simple arrangement where all devices send their interrupt-requests over a single control line in the bus.
- When the processor receives an interrupt request over this control line, how does it know which device is requesting an interrupt?
- This information is available in the status register of the device requesting an interrupt (KIRQ- DIRQ)
  - The status register of each device has an *IRQ* bit which it sets to 1 when it requests an interrupt.
- Interrupt service routine can poll the I/O devices connected to the bus. The first device with *IRQ* equal to 1 is the one that is serviced.(polling mechanism definition)
- Polling mechanism is easy, but time consuming to query the status bits of all the I/O devices connected to the bus.

# Handling Multiple I/O devices

## Vectored interrupts

- The device requesting an interrupt may identify itself directly to the processor.
  - Device can do so by sending a special code (4 to 8 bits) the processor over the bus.
  - Code supplied by the device may represent a part of the starting address of the interrupt-service routine.
  - The remainder of the starting address is obtained by the processor based on other information such as the range of memory addresses where interrupt service routines are located.
- Usually the location pointed to by the interrupting device is used to store the starting address of the interrupt-service routine(interrupt vector).

# Handling Multiple I/O devices

- Previously, before the processor started executing the interrupt service routine for a device, it disabled the interrupts from the device.
- In general, same arrangement is used when multiple devices can send interrupt requests to the processor.
  - During the execution of an interrupt service routine of device, the processor does not accept interrupt requests from any other device.
  - Since the interrupt service routines are usually short, the delay that this causes is generally acceptable.
- However, for certain devices this delay may not be acceptable.
  - Which devices can be allowed to interrupt a processor when it is executing an interrupt service routine of another device?

# Handling Multiple I/O devices

- Nested Interrupts-Interrupt proirity
- I/O devices are organized in a priority structure:
  - An interrupt request from a high-priority device is accepted while the processor is executing the interrupt service routine of a low priority device.
- A priority level is assigned to a processor that can be changed under program control.
  - Priority level of a processor is the priority of the program that is currently being executed.
  - When the processor starts executing the interrupt service routine of a device, its priority is raised to that of the device.
  - If the device sending an interrupt request has a higher priority than the processor, the processor accepts the interrupt request.

# Handling Multiple I/O devices

- Processor's priority is encoded in a few bits of the processor status register.
  - Priority can be changed by instructions that write into the processor status register.
  - Usually, these are privileged instructions, or instructions that can be executed only in the supervisor mode.
  - Privileged instructions cannot be executed in the user mode.
  - Prevents a user program from accidentally or intentionally changing the priority of the processor.
- If there is an attempt to execute a privileged instruction in the user mode, it causes a special type of interrupt called as privilege exception.

# Multiple priority scheme

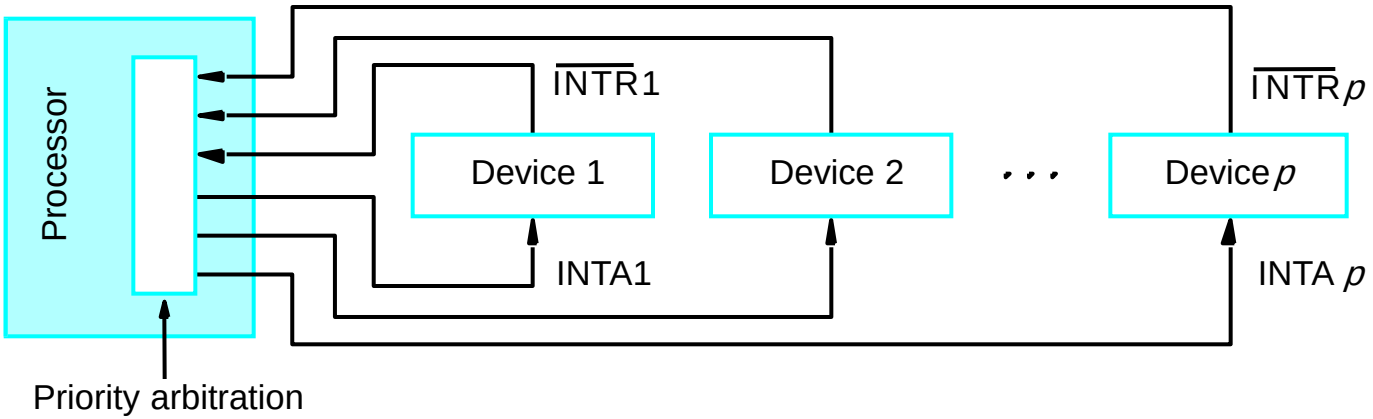



**Figure 4.7** Implementation of interrupt priority using individual interrupt-request and acknowledge lines.

- *Each device has a separate interrupt-request and interrupt-acknowledge line.*
- *Each interrupt-request line is assigned a different priority level.*
- *Interrupt requests received over these lines are sent to a priority arbitration circuit in the processor.*
- *If the interrupt request has a higher priority level than the priority of the processor, then the request is accepted.*

# Handling Multiple I/O devices

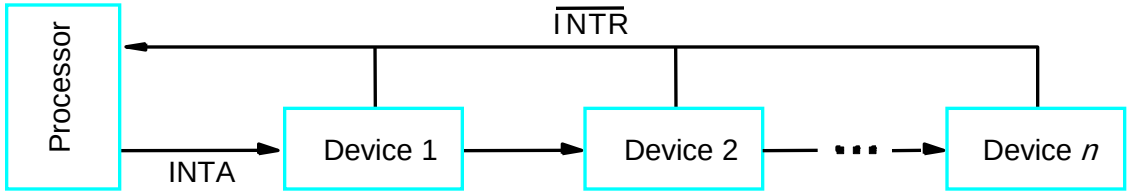
- Which interrupt request does the processor accept if it receives interrupt requests from two or more devices simultaneously?.
- If the I/O devices are organized in a priority structure, the processor accepts the interrupt request from a device with higher priority.
  - Each device has its own interrupt request and interrupt acknowledge line.
  - A different priority level is assigned to the interrupt request line of each device.
- However, if the devices share an interrupt request line, then how does the processor decide which interrupt request to accept?

# Handling Multiple I/O devices with proirity

***1)Polling scheme:***

- *If the processor uses a polling mechanism to poll the status registers of I/O devices to determine which device is requesting an interrupt.*
- *In this case the priority is determined by the order in which the devices are polled.*
- *The first device with status bit set to 1 is the device whose interrupt request is accepted.*

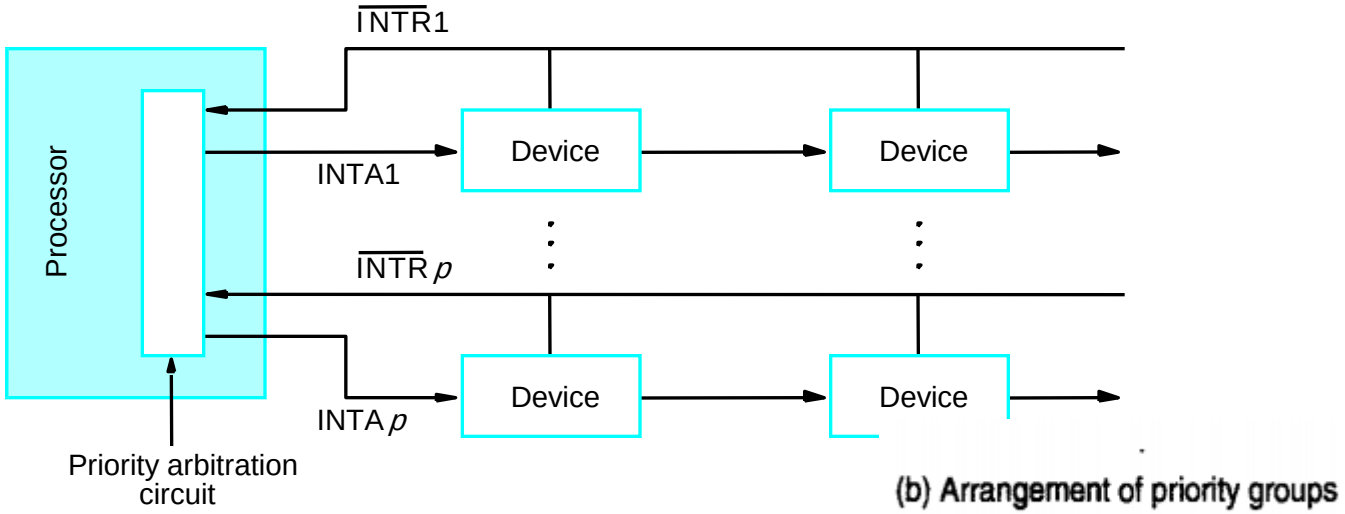
***2) Daisy chain scheme:***




- *Devices are connected to form a daisy chain.*
- *Devices share the interrupt-request line, and interrupt-acknowledge line is connected to form a daisy chain.*
- *When devices raise an interrupt request, the interrupt-request line is activated.*
- *The processor in response activates interrupt-acknowledge(INTA).*
- *Received by device 1, if device 1 does not need service, it passes the signal to device 2.*
- *Device that is electrically closest to the processor has the highest priority.*

# Handling Multiple I/O devices

- *When I/O devices were organized into a priority structure, each device had its own interrupt-request and interrupt-acknowledge line.*
- *When I/O devices were organized in a daisy chain fashion, the devices shared an interrupt-request line, and the interrupt-acknowledge propagated through the devices.*
- *A combination of priority structure and daisy chain scheme can also used.*




(b) Arrangement of priority groups

- *Devices are organized into groups.*
- *Each group is assigned a different priority level.*
- *All the devices within a single group share an interrupt-request line, and are connected to form a daisy chain.*

# Controlling device requests

- Only those devices that are being used in a program should be allowed to generate interrupt requests.
- To control which devices are allowed to generate interrupt requests, the interface circuit of each I/O device has an interrupt-enable bit.
  - If the interrupt-enable bit in the device interface is set to 1, then the device is allowed to generate an interrupt-request.
- Interrupt-enable bit in the device's interface circuit determines whether the device is allowed to generate an interrupt request.
- Interrupt-enable bit in the processor status register or the priority structure of the interrupts determines whether a given interrupt will be accepted.

# Controlling device requests

The control needed is usually provided in the form of an interrupt-enable bit in the device's interface circuit. The keyboard interrupt-enable, KEN, and display interrupt-enable, DEN, flags in register CONTROL in Figure 4.3 perform this function. If either of these flags is set, the interface circuit generates an interrupt request whenever the corresponding status flag in register STATUS is set. At the same time, the interface circuit sets bit KIRQ or DIRQ to indicate that the keyboard or display unit, respectively, is requesting an interrupt. If an interrupt-enable bit is equal to 0, the interface circuit will not generate an interrupt request, regardless of the state of the status flag.

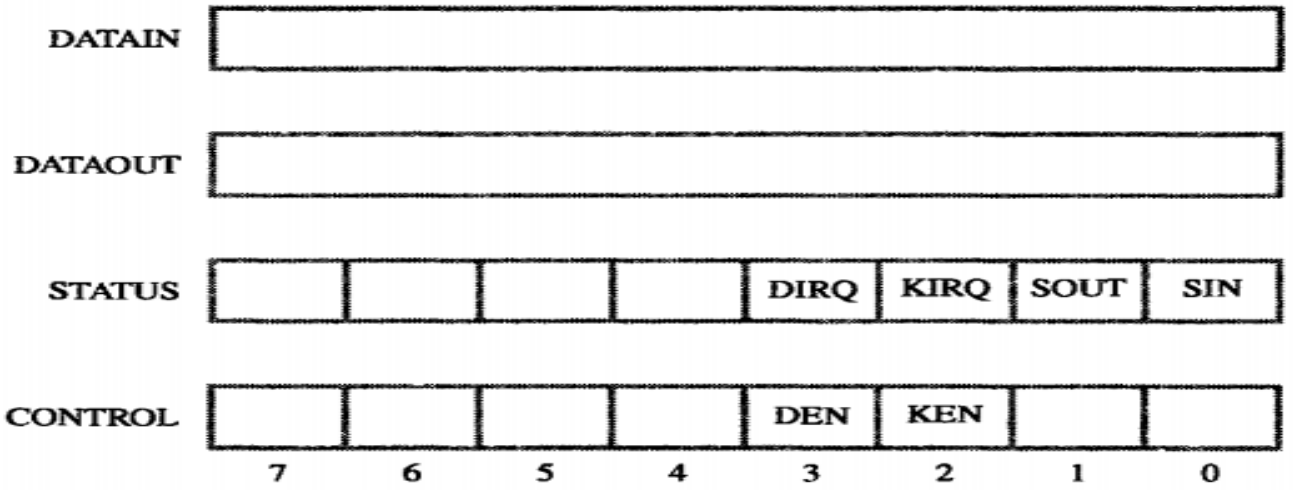



**Figure 4.3** Registers in keyboard and display interfaces.

# Exceptions

- Interrupts caused by interrupt-requests sent by I/O devices.
- Interrupts could be used in many other situations where the execution of one program needs to be suspended and execution of another program needs to be started.
- In general, the term exception is used to refer to any event that causes an interruption.
  - Interrupt-requests from I/O devices is one type of an exception.
- Other types of exceptions are:
  - Recovery from errors
  - Debugging
  - Privilege exception

# Exceptions (contd..)

- Many sources of errors in a processor. For example:
  - Error in the data stored.
  - Error during the execution of an instruction.
- When such errors are detected, exception processing is initiated.
  - Processor takes the same steps as in the case of I/O interrupt-request.
  - It suspends the execution of the current program, and starts executing an exception-service routine.
- Difference between handling I/O interrupt-request and handling exceptions due to errors:
  - In case of I/O interrupt-request, the processor usually completes the execution of an instruction in progress before branching to the interrupt-service routine.
  - In case of exception processing however, the execution of an instruction in progress usually cannot be completed.

# Exceptions (contd..)

- Debugger uses exceptions to provide important features:
  - Trace,
  - Breakpoints.
- Trace mode:
  - Exception occurs after the execution of every instruction.
  - Debugging program is used as the exception-service routine.
- Breakpoints:
  - Exception occurs only at specific points selected by the user.
  - Debugging program is used as the exception-service routine.

# Exceptions (contd..)

- Certain instructions can be executed only when the processor is in the supervisor mode. These are called privileged instructions.
- If an attempt is made to execute a privileged instruction in the user mode, a privilege exception occurs.
- Privilege exception causes:
  - Processor to switch to the supervisor mode,
  - Execution of an appropriate exception-servicing routine.

# Direct Memory Access

- Program-controlled I/O:
  - Processor polls a status flag in the device interface.
  - Overhead associated with polling the status flag.
- I/O using interrupts:
  - Processor waits for the device to send an interrupt request.
  - Overhead associated with saving and restoring the Program Counter (PC) and other state information.
- In addition, if we want to transfer a group of words, the processor needs to execute instructions which increment the memory address and keep track of the word count.
- To transfer large blocks of data at high speed, an alternative approach is used.

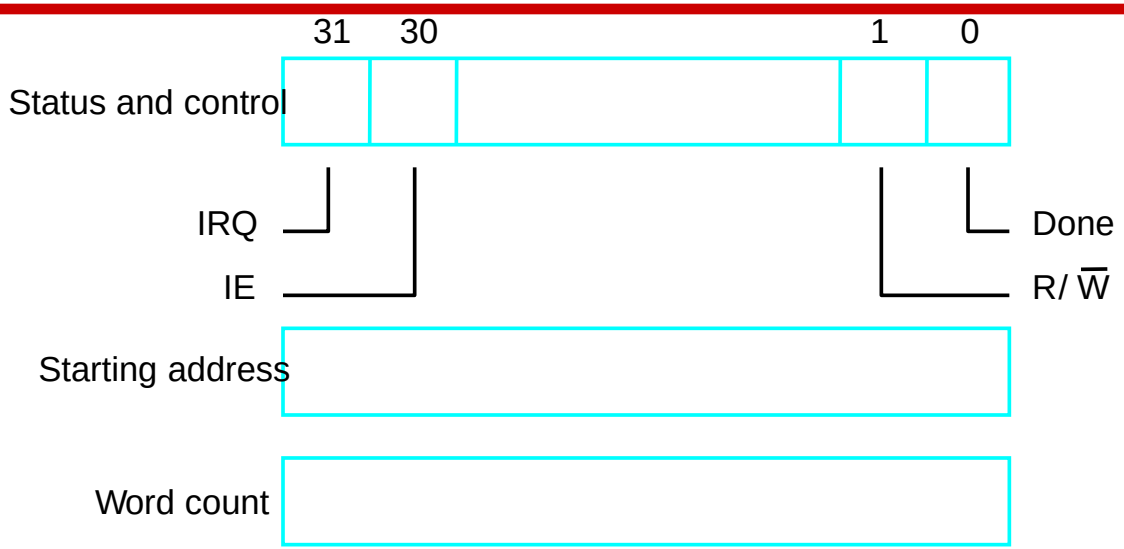
# Direct Memory Access (contd..)

- Direct Memory Access (DMA):
  - A special control unit may be provided to transfer a block of data directly between an I/O device and the main memory, without continuous intervention by the processor.
- Control unit which performs these transfers is a part of the I/O device's interface circuit. This control unit is called as a DMA controller.
- DMA controller performs functions that would be normally carried out by the processor:
  - For each word, it provides the memory address and all the control signals.
  - To transfer a block of data, it increments the memory addresses and keeps track of the number of transfers.

# Direct Memory Access (contd..)

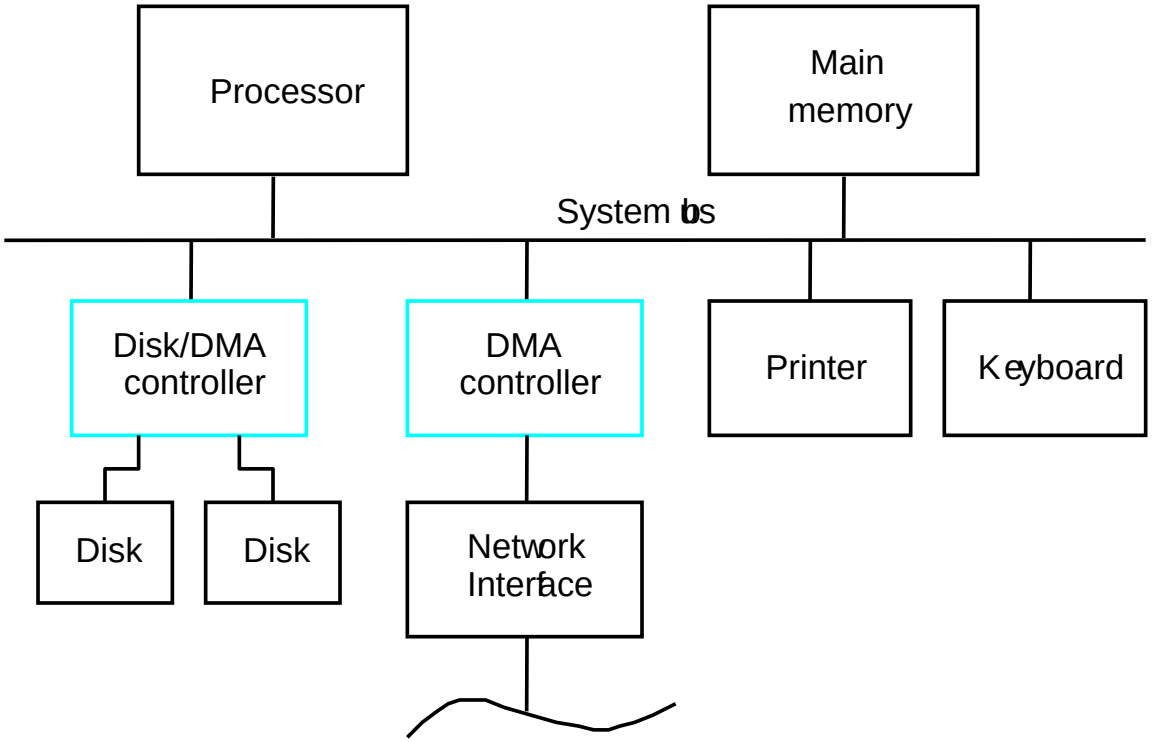
- DMA controller can transfer a block of data from an external device to the processor, without any intervention from the processor.
  - However, the operation of the DMA controller must be under the control of a program executed by the processor. That is, the processor must initiate the DMA transfer.
- To initiate the DMA transfer, the processor informs the DMA controller of:
  - Starting address,
  - Number of words in the block.
  - Direction of transfer (I/O device to the memory, or memory to the I/O device).
- Once the DMA controller completes the DMA transfer, it informs the processor by raising an interrupt signal.

# Direct Memory Access (contd..)




- *DMA controllers have one register to hold the starting address, and one register to hold the word count.*
- *A third register called as status register contains status and control bits.*
- *R/W = 1 specifies a read operation, R/W = 0 specifies a write operation.*
- *When the transfer is complete, the Done bit is set to 1.*
- *If IE = 1, the DMA controller raises an interrupt after the transfer is complete.*
- *To raise an interrupt, it sets the IRQ bit to 1.*
- *Status register may also record other information such as whether the transfer took place correctly, or errors occurred.*

# Direct Memory Access (contd..)




- *DMA controller connects a high-speed network to the computer bus.*
- *Disk controller, which controls two disks also has DMA capability. It provides two DMA channels.*
- *It can perform two independent DMA operations, as if each disk has its own DMA controller. The registers to store the memory address, word count and status and control information are duplicated.*

# Direct Memory Access (contd..)

- Processor and DMA controllers have to use the bus in an interwoven fashion to access the memory.
  - DMA devices are given higher priority than the processor to access the bus.
  - Among different DMA devices, high priority is given to high-speed peripherals such as a disk or a graphics display device.
- Processor originates most memory access cycles on the bus.
  - DMA controller can be said to "steal" memory access cycles from the bus. This interweaving technique is called as "cycle stealing".
- An alternate approach is to provide a DMA controller an exclusive capability to initiate transfers on the bus, and hence exclusive access to the main memory. This is known as the block or burst mode.